# Retinex in Matlab

***Brian Funt and Florian Ciurea***
***School of Computing Science, Simon Fraser University***
***Burnaby, British Columbia, Canada***


***John McCann***
***McCann Imaging***
***Belmont, Massachusetts***

## Abstract

Many different descriptions of retinex methods of lightness computation exist. This paper provides concise MATLAB implementations of two of the spatial techniques of making pixel comparisons. The code is presented along with test results on several images and a discussion of the results**.** The paper also discusses the calibration of input images and the post-retinex processing required to display the output images.

## Introduction

The retinex model[1] for the computation of lightness was introduced by Land and McCann.[9] Since that time Land and his colleagues have described several variants on the original method.[16,12,5,13,20] The variants on retinex mainly aim to improve the computational efficiency of the model while preserving its basic underlying principles.

Retinex calculations aim to calculate the sensory response of lightness. It is important to distinguish between physical reflectance, the sensation of lightness, and perceived reflectance, which are three distinct entities. A single model can attempt to calculate only one of the three—the retinex goal is to calculate the sensation of lightness. Consider the case of two faces of a white cube, one in direct sunlight and the other in shadow. Physical reflectance is a measure of a property of the cube's surface relating its radiance to its irradiance.

The reflectances of the two faces are identical. Sensations, on the other hand, are the appearances of the faces of the cube in the sun and the shade. To create the same appearances in a painting, a fine arts painter would mix white with a little yellow to make the sunny face, but use white with blue and a little black to reproduce the appearance of the shadowed face. These samples of different colored paints are measures of sensation. Here the two faces are different.[17] In comparison, the question of the perceived reflectances of the cube's surfaces involves cognition. It asks the observer to recognize the paint on the cube. Asked to repaint the cube, the observer is not confused by sun and shade, and would simply apply white paint. In terms of perception, the two faces of the cube are identical. In contrast, retinex calculates lightness sensations—it cannot be used to calculate physical reflectances or perceived reflectances.

The first model designed to calculate lightness was described in Land's Ives Medal Address to the Optical Society of America in 1968 and later published.[9] This lecture included a working demonstration of a primitive electronic retinex camera. This was followed by publications and patents with additional details and improvements.[5,10,11] McCann McKee and Taylor[16] described a study of human color constancy that included color-matching experiments, the details of the lightness model and successful results of modeling the experimental data. This result was further developed to show that there is no effect of cone pigment adaptation in color constancy.[19] The retinex operators were selected for simplicity to mimic biological operators that sum, difference and rectify input signals to obtain spatial interactions.

Dynamic range compression of real images was described in a patent by Frankle and McCann.[5] This implementation used specialized hardware (International Imaging Systems I²S image processor with scrollable 8-bit image planes) for efficient image calculation. It described the idea that information from $2^n$ pixels is accumulated after n steps of the process. This patent also described the multiresolution approach to retinex calculation used for computer applications.[24,20]

## Appropriate Input Data

For testing the retinex model it is crucial that the data be calibrated in the sense that the image digit values must be a logarithmic function of scene radiance and they must be represented with sufficient precision. McCann[18] used a slope 1.0 photographic film to capture real images (Ektachrome 5071 slide duplicating film). He was able to measure an in-camera dynamic range of 3.5 log units. The

---

[1]*McCann [20] refers to these models as Ratio-Product-Reset-Average, but for simplicity here we call these operations the retinex model. Frankle and McCann [5] provide complete FORTRAN code for their algorithm with extensive discussion of image processing steps that follow spatial comparisons.*

importance of the logarithmic function follows from Wallach's experiments on appearance.[23] He showed that equal radiance ratios generate equal lightness differences. A pair of papers, i.e., a 20% gray paper and a 100% white paper, have the same lightness difference in sun and shade. The pair also has a $\log_{10}$ edge difference of 0.7, regardless of illumination. If the input image data deviates from logarithmic, then the $\log_{10}$ edge difference for these papers will change with illumination, and the calculated lightness difference of the pair will change. For retinex to work well, edge ratios, or $\log_{10}$ differences, within an object must be independent of illumination. Accurate logarithmic calibration guarantees this to be the case.

The need for sufficient precision can be demonstrated by comparing two routes to the same scaling of an image. In one, we convert raw data to 8-bit $\log_{10}$ digits. This represents the image well. In the other, we convert raw data to 8-bit linear and to 8-bit $\log_{10}$. The 8-bit linear stage truncates the information severely. Where the first 8-bit $\log_{10}$ processing assigns 82 digits to image shades between light gray and black, the second linear 8-bit processing assigns 1 digit (See Table 1). One cannot take an existing 8-bit image, apply a log to it and have meaningful image data for testing the retinex model.

Nevertheless, retinex often enhances random images that have unknown and unknowable radiances for inputs.[5,1,14] The process improves the visibility of dark objects while maintaining the visual discrimination of the light areas. Unlike lookup tables, which improve one range of radiance at the expense of others, retinex improves visual differentiation in all ranges of radiances. The danger is that artifacts such as noise create artificial edge information that is enhanced by retinex processing. The ability to bring out shadow detail is limited by image noise.

## Retinex Operators

The original Land and McCann paper[9] described four steps for each iteration of a retinex calculation: ratio, product, reset and average. With the exception of reset[13] these operators have remained the same over the years. These operators are iteratively applied to an image, but the manner in which they are applied has varied. The focus of this paper is to list specific details of how these four operators are applied to the image.

A fundamental concept behind retinex computation of lightness at a given image pixel is the comparison of the pixel's value to that of other pixels. The main difference between the retinex algorithms is the way in which the other comparison pixels are chosen, including the order in which they are chosen. They use the same calculations but have dramatically different computational efficiencies in dealing with large real images. The original way of defining comparisons is by following a path, or set of paths, from pixel to neighboring pixel through the image.[9] Lightness estimates are accumulated along the path in a "sequential product" SP. SP starts as 1 and then is modified by multiplying it with the ratio of the next pair of pixels

along the path. In the case of path following, path length affects the results substantially. Short paths mean the comparison is made only to others in a spatially localized group of pixels. Intermediate path lengths are to be used when modeling human vision. Infinite path lengths result in a degenerate case in which the output image is simply a scaled version of the input image. Infinite path lengths should not be used to model vision.[2]

A "reset" step is a second important feature of retinex. Each time a comparison is made the SP is tested; if it exceeds 1.0, it is reset to 1.0. In this case, the value 1.0 becomes the current lightness estimate. A third aspect of retinex is the way in which lightness estimates obtained from different paths to a pixel are combined. In earlier versions, retinex also included a "thresholding" step; however, it is not included in later versions[20] and is not part of the Matlab implementations shown below. The fourth step averages present values of the Product with previous ones.

## Implementations

We have chosen two versions of retinex to implement. The first is a computer-based version described by McCann,[20] which we will refer to as McCann99 retinex. The second is an older specialized-hardware version,[5] which we will call Frankle-McCann retinex. The two versions both replace path following with more computationally efficient spatial comparisons. McCann99 retinex creates a multi-resolution pyramid from the input by averaging image data. It begins the pixel comparisons at the most highly averaged, or top level of the pyramid. After computing lightness on the image at a reduced resolution, the resulting lightness values are propagated down, by pixel replication, to the pyramid's next level as initial lightness estimates at that level. Further pixel comparisons refine the lightness estimates at the higher resolution level and then those new lightness estimates are again propagated down a level in the pyramid. This process continues until New Products have been computed for the pyramid's bottom level.

In comparison, Frankle-McCann retinex uses single pixel comparisons with variable separations. An important difference between this method and that described in Land and McCann[9] is that there are no paths. A single pixel eventually averages different products from all other pixels. The advantage of this structure, and also for the multi-resolution approach, is that long distance interactions are propagated with fewer comparisons.

### McCann99 Multi-Level Retinex Details

For this implementation the input images must be of dimension $w \cdot 2^n$ by $h \cdot 2^n$ where $w \geq h$ and $w$ and $h$ are integers in the range $1 \leq w, h \leq 5$. This constraint arises from the fact that each level of the image pyramid differs from previous levels by a factor of 2 in each dimension. It is not a serious limitation in practice.

The algorithm assumes that input digits are proportional to the logarithm of scene radiance and are of

meaningful precision. Using logarithms simplifies the computation of radiance ratios, which become simple differences. It also implies that when results from different spatial comparisons are averaged, the averaging is in log space and hence equivalent to a geometric mean.

In the first step, the log image is averaged down to the lowest resolution level, which depending on the input dimensions will be of the size 1×1, 1×2, 1×3, 2×3, 3×4, 3×5, 4×5 or 5×5. At each step the resolution level will be doubled. The number of layers in the pyramid depends on the size of the input image. The number of layers will be the greatest power of 2 dividing both the width and height of the input images as calculated by the function ComputeSteps.

When the results (called "New Products") at one level of dimension n-by-m have been computed, the values are then replicated to form an "Old Product" image of dimension 2n-by-2m. In our implementation, we pad the Old Product image with zeroes to simplify handling boundary conditions. These extra pixels are discarded at the end of the computation.

At all levels the New Product, a precursor of calculated lightness, for each pixel is computed by visiting each of its 8 immediately neighboring pixels in clockwise order. Each visit involves a ratio-product-reset-average operation,[20] which is implemented by the function CompareWithNeighbor. It subtracts the neighbor's log luminance (the ratio step) and then adds the result to the old product (the product step). If the result exceeds the maximum defined by Maximum, it is reset to Maximum (the reset step). Finally, the new product for the pixel

obtained by comparison to its neighbor is averaged with the previous old product.

A crucial parameter to the McCann99 algorithm is the number of times a pixel's neighbors are to be visited. In the code, this is set by nIterations. It controls the number of times the neighbors are cycled through, which as a result affects the distance at which pixels influence one another. This occurs because the New Product values for all pixels are being computed in parallel, so that after one iteration all neighboring pixels have had their New Products values updated. Hence, in the second iteration these new values involve information propagated from beyond a pixel's immediate neighbors. In the limiting case of an infinite number of iterations, the algorithm converges to produce an output image that is simply the input image scaled by the image's maximum value. A practical value for the number of iterations is 4. The final step is to scale the New Product values to make an estimated lightness (see Section "Scaling of retinex output to desired media and purpose"). In the case of color images, the function retinex_mccann99 must be applied to each of the color channels independently.

The code is based on MATLAB 5 (Version 5.1.0.421). For the reader unfamiliar with Matlab, the statement IP (IP > Maximum) = Maximum, which sets all values in matrix IP that are greater than Maximum to Maximum, demonstrates an important feature of the language; namely, that most of the functions and operators work on whole matrices applying the given function to all matrix elements.

**Matlab implementation of the McCann99 retinex**

**Notation:**

| | |
|---|---|
| L: | logarithmic input image |
| nIterations: | number of iterations for each pixel |
| nLayers: | number of pyramid layers |
| OP: | matrix of Old Products for all pixels |
| RR: | input radiance |
| NP: | matrix of New Products for all pixels |
| OPE: | OP padded with zeros for doing all computations at once |
| RRE: | RR padded with two additional columns and two additional rows |
| IP: | Intermediate Product computed with the Ratio-Product |

```
function Retinex = retinex_mccann99(L, nIterations)
global OPE RRE Maximum
[nrows ncols] = size(L);                                % get size of the input image
nLayers = ComputeLayers(nrows, ncols);                  % compute the number of pyramid layers
nrows = nrows/(2^nLayers);                               % size of image to process for layer
0
ncols = ncols/(2^nLayers);
if (nrows*ncols > 25)                                    % not processing images of area > 25
  error('invalid image size.')                          % at first layer
end
Maximum = max(L(:));                                     % maximum color value in the image
OP = Maximum*ones([nrows ncols]);                        % initialize Old Product
for layer = 0:nLayers
   RR = ImageDownResolution(L, 2^(nLayers-layer));       % reduce input to required layer size

   OPE = [zeros(nrows,1) OP zeros(nrows,1)];             % pad OP with additional columns
   OPE = [zeros(1,ncols+2); OPE; zeros(1,ncols+2)];      % and rows
   RRE = [RR(:,1) RR RR(:,end)];                         % pad RR with additional columns
   RRE = [RRE(1,:); RRE; RRE(end,:)];                    % and rows

   for iter = 1:nIterations
     CompareWithNeighbor(-1, 0);                         % North
     CompareWithNeighbor(-1, 1);                         % North-East
     CompareWithNeighbor(0, 1);                          % East
     CompareWithNeighbor(1, 1);                          % South-East
     CompareWithNeighbor(1, 0);                          % South
     CompareWithNeighbor(1, -1);                         % South-West
     CompareWithNeighbor(0, -1);                         % West
     CompareWithNeighbor(-1, -1);                          % North-West
   end
   NP = OPE(2:(end-1), 2:(end-1));
   OP = NP(:, [fix(1:0.5:ncols) ncols]);                    %%% these two lines are equivalent
with
   OP = OP([fix(1:0.5:nrows) nrows], :);                    %%% OP = imresize(NP, 2) if using
Image
   nrows = 2*nrows; ncols = 2*ncols;                     % Processing Toolbox in MATLAB
end
Retinex = NP;

function CompareWithNeighbor(dif_row, dif_col)
global OPE RRE Maximum

% Ratio-Product operation
IP = OPE(2+dif_row:(end-1+dif_row), 2+dif_col:(end-1+dif_col)) + ...
     RRE(2:(end-1),2:(end-1)) - RRE(2+dif_row:(end-1+dif_row), 2+dif_col:(end-1+dif_col));

IP(IP > Maximum) = Maximum;                              % The Reset step

% ignore the results obtained in the rows or columns for which the neighbors are undefined
if (dif_col == -1) IP(:,1) = OPE(2:(end-1),2); end
if (dif_col == +1) IP(:,end) = OPE(2:(end-1),end-1); end
if (dif_row == -1) IP(1,:) = OPE(2, 2:(end-1)); end
if (dif_row == +1) IP(end,:) = OPE(end-1, 2:(end-1)); end
NP = (OPE(2:(end-1),2:(end-1)) + IP)/2;                  % The Averaging operation
OPE(2:(end-1), 2:(end-1)) = NP;

function Layers = ComputeLayers(nrows, ncols)
power = 2^fix(log2(gcd(nrows, ncols)));                  %  start  from  the  Greatest  Common
Divisor
while(power > 1 & ((rem(nrows, power) ~= 0) | (rem(ncols, power) ~= 0)))
   power = power/2;                                      % and find the greatest common divisor
end                                                      % that is a power of 2
Layers = log2(power);
```

```
function Result = ImageDownResolution(A, blocksize)
[rows, cols] = size(A);                                 % the input matrix A is viewed as
result_rows = rows/blocksize;                           % a series of square blocks
result_cols = cols/blocksize;                           % of size = blocksize
Result = zeros([result_rows result_cols]);
for crt_row = 1:result_rows                             % then each pixel is computed as

   for crt_col = 1:result_cols                          % the average of each such block
      Result(crt_row, crt_col) = mean2(A(1+(crt_row-1)*blocksize:crt_row*blocksize, ...
                                     1+(crt_col-1)*blocksize:crt_col*blocksize));
   end
end
```

### Frankle-McCann Retinex

Like McCann99 retinex, Frankle-McCann retinex computes long-distance interactions between pixels first and then progressively moves to short-distance interactions. In Frankle-McCann, the spacing between the pixels being compared decreases with each step. The direction between pixels also changes at each step, in clockwise order. At each step, the comparison is implemented using the Ratio-Product-Reset-Average operation. The process continues until the spacing decreases to 1 pixel.

The original algorithm assumed the input image to be 512x512. This followed the hardware design of the $I^2S$. As a result, the initial spacing between pixels started at 256.

We have generalized the algorithm slightly so that our implementation will work on an image of arbitrary size. In this case, the initial spacing (as encoded by the variable 'shift') is computed as the largest power of 2 smaller than both of the input image dimensions.

The function CompareWith(s_row, s_col) updates the current lightness estimate for a pixel using the ratio-product-reset-average operation described above. In the case of McCann99, it is based on the pixel located at a distance of s_row, s_col. The square spiral path structure in this implementation means that when this function is called, one of the two parameters will always be zero. The original Frankle and McCann[5] implementation had the option of either square or 8-direction comparisons.

### Matlab Implementation of Frankle-McCann retinex

```
function Retinex = retinex_frankle_mccann(L, nIterations)
global RR IP OP NP Maximum
RR = L;
Maximum = max(L(:));                                    % maximum color value in the image
[nrows, ncols] = size(L);

shift = 2^(fix(log2(min(nrows, ncols))))-1;             % initial shift
OP = Maximum*ones(nrows, ncols)                         % initialize Old Product

while (abs(shift) >= 1)
   for i = 1:nIterations
      CompareWith(0, shift);                            % horizontal step
      CompareWith(shift, 0);                            % vertical step
   end
   shift = -shift/2;                                     % update the shift
end
Retinex = NP;

function CompareWith(s_row, s_col)
global RR IP OP NP Maximum
IP = OP;
if (s_row + s_col > 0)
   IP((s_row+1):end, (s_col+1):end) = OP(1:(end-s_row), 1:(end-s_col)) + ...
   RR((s_row+1):end, (s_col+1):end) - RR(1:(end-s_row), 1:(end-s_col));
else
   IP(1:(end+s_row), 1:(end+s_col)) = OP((1-s_row):end, (1-s_col):end) + ...
   RR(1:(end+s_row),1:(end+s_col)) - RR((1-s_row):end, (1-s_col):end);
end
IP(IP > Maximum) = Maximum;                             % The Reset operation
NP = (IP + OP)/2;                                       % average with the previous Old Product
OP = NP;                                                % get ready for the next comparison
```

# Retinex Parameters

All spatial operators use variable parameters to appropriately match their effects to input images. For example, this is true of unsharp masking, jpeg and retinex spatial operators.

The purpose of unsharp masking is to change the spatial content in the image, particularly in the high-spatial-frequency components. When successfully used the image looks sharper and free of artifacts. With inappropriate parameters the process will generate artifacts that are visible to the observer. If we compare the effects of a particular unsharp mask on same-size prints of a 256-by-256 digital image with the effects on an 2k-by-2k image, we see that they act very differently. A sharpening filter that is appropriate for the small image will have no effect on large images, while an appropriate filter for the large images will introduce artifacts in small ones. Given a print size and a viewing distance, one can optimize the shape of the filter kernel. The choice of sharpening kernel is selected so as to keep artifacts below visual threshold, which is a function of both spatial frequency,[3] size of the display[15] and light intensity of the display.[22]

An analogous spatial dependence is found in jpeg compression where knowledge of human sensitivity to spatial information is used to reduce the number of bits for rendering a visually similar image.[21] When we select a quality factor, we are controlling an underlying array of coefficients that filter the data so as to reduce the data needed to recreate the image. To make two same-size prints from a 256-by-256 versus a 2k-by-2k image, requires different jpeg coefficients. Any reduction in information will likely be visible in the small number-of-pixel image, while the larger image might well be compressed by factors of 10:1 or 20:1 without noticeable effect. The difference arises because the size and viewing distance control what information the observer can see in the final prints. Large digital files often contain more information than can be seen in a small print. This is the information that jpeg discards. As with unsharp masking the user specifies the spatial parameters to optimize performance and avoid artifacts.

Retinex has parameters that are responsive to both spatial frequency and dynamic range of the input data. The number of iterations, as specified in the Matlab code by 'nIterations', controls the amount of dynamic range compression and sets the stage for a different level of post-processing by a postlut. The term "postlut" derives from historical use of image processing hardware using a lookup table. Postlut processing simply refers to the application of a function f applied uniformly to every image pixel, $I(x,y)=f(I(x,y))$, for all image locations $(x,y)$. The effect of the number of iterations can be seen in Figure 1.

As we can see the effect of the number of iterations is to reduce the contrast of the images as demonstrated by the smaller range in the histograms. The process moves the entire image into a smaller dynamic range, with smaller digit differences representing edge ratios. With very few iterations, the range of output digits is small. The postlut expansion (stretching of the image intensities) must be large to regenerate edge ratios appropriate for a print. With more iterations the range of output digits is larger. The postlut expansion will be moderate to regenerate edge ratios. With a very large number of iterations the range of output digits is large, approaching that of the input image. The postlut expansion must be small to none to regenerate edge ratios. The amount of postlut expansion will vary with the amount of dynamic range compression.

The examples of unsharp masking and jpeg compression demonstrated the need for selecting the right parameters to match viewing size and viewing distance. Analogously, the viewing distance, the viewing size, the dynamic range and noise level of the input image, the number of iterations, and the postlut are all important to make artifact-free retinex images.

## Scaling of Retinex Output to Desired Media and Purpose

As shown in Figure 1, the contrast of the output is controlled by the number of iterations. This parameter can vary the output from radical to no dynamic range compression. The input data also plays a major role. The total dynamic range of input data determines the magnitude of radiance ratio associated with each digit. The final parameter is the postlut that matches the final new product with the output media. That media can be a printer, a monitor, a LCD display, a system profile, a 3-D plot of output at each pixel (output equal height), a pseudocolor image. The essential idea is that the input calibration controls the correlation between digital differences and radiances in the world. The number of iterations controls the degree of compression. The postlut controls the rendition of New Product digital differences in the output media. All three parameters are crucial to the process. All three share the control of the output image. They can be used only as well designed sets. They are not randomly interchangeable.

## Results on Test Images

Figures 2 through 5 illustrate the behavior of the two algorithms. Figure 2 shows the behavior when the input is a simple square at the very center of the image. A slight asymmetry can be seen in both the McCann99 (using 4 iterations comparing 8 nearest neighbors) and Frankle-McCann (using 4 iterations of 4 directions) outputs. The McCann99 output shows the effect of processing the 8 pixel neighbors in clockwise order. No postlut has been applied to these images.
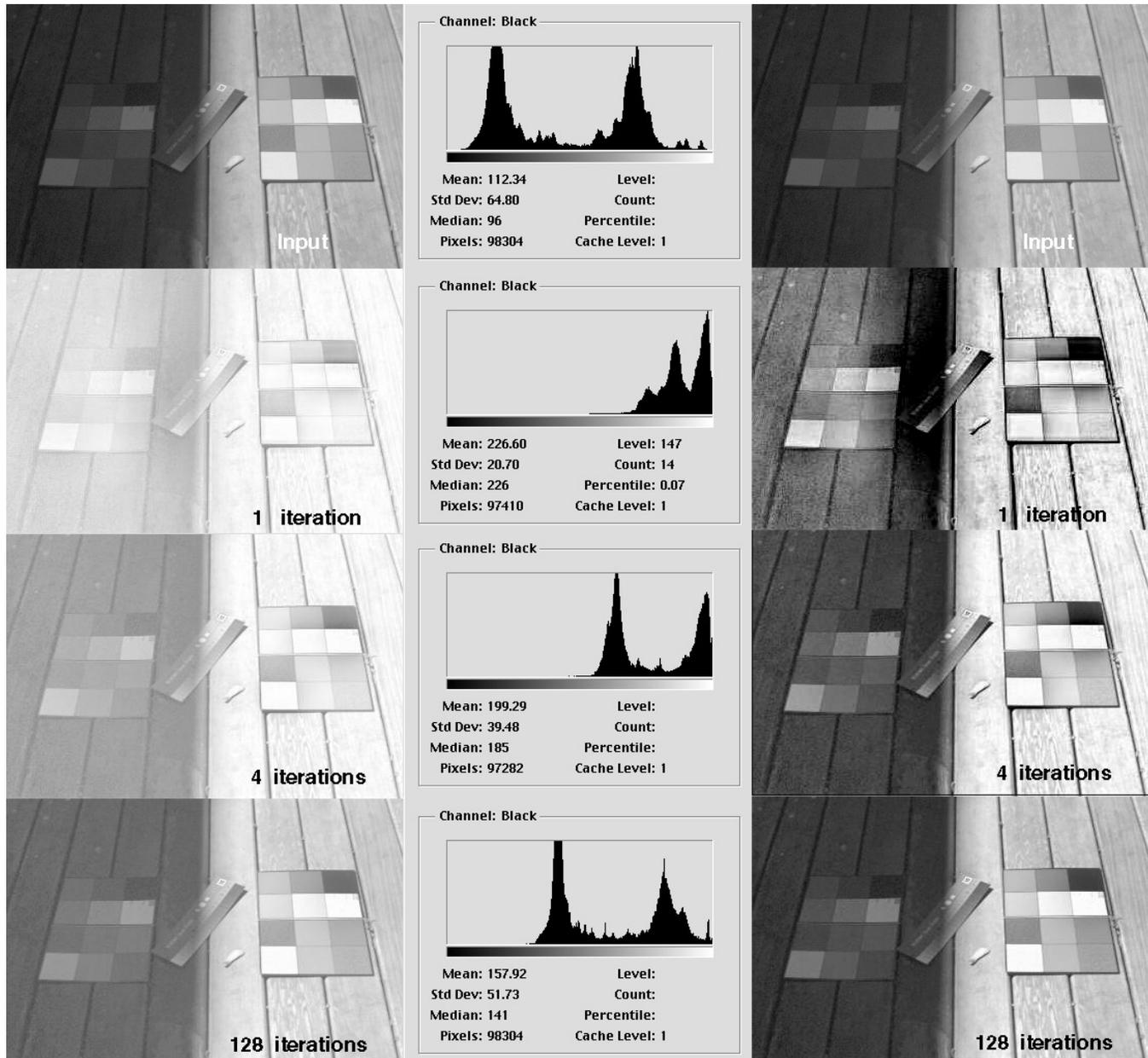
*Figure 1 demonstrates the role of number of iterations and postluts. The first column shows the effect of spatial comparisons (ratio-product-reset-average). The second column is the histogram of the images in the first column. The third column show images that have been stretched by a different postlut for each number of iterations. The first row shows the input $log_{10}$ image scaled so that 3.5 $log_{10}$ units covers 0-255. The sun half of the image is on the right and the shade half is on the left. The shade image is a lower radiance copy of the sun image. The histogram of this image is in the second column. The third column image is the same as first column, illustrating that it has a slope 1.0 postlut. Output equals input. The second row shows an output image using one iteration, with its histogram. Here the output dynamic range has been compressed into the top 25% of the 0-255 digit range. A slope 4.0 linear postlut will stretch the first column image to render contrast in the sun properly. It is very steep and generates artifacts. The third row shows the output for four iterations, and its histogram. Here the range data has been compressed from 3 log units to 1.5. A slope 2.0 postlut has only to expand the data from 128 to 0. The fourth row shows the output for 128 iterations and its histogram. There is only a 25% compression. A slope 1.5 postlut will be very gentle; however, the improvement of the shadow detail in the third column output image is minimal. In this figure we used simple linear postluts to illustrate how calibration, number of iterations and postlut work together. To optimize the image these postluts should be shaped so as to take into account the response of the output device and the tone reproduction curve desired. (See Appendix II & III of Frankle and McCann for details[5]).*
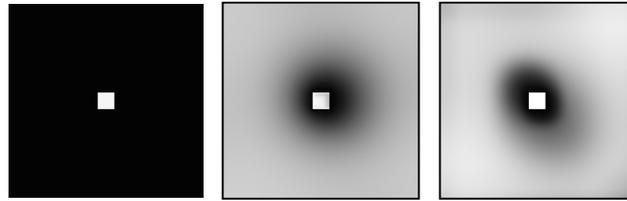
*Figure 2. Effect of McCann99 processing on input of a single bright square against a black background. In the limiting case of the square being a single pixel, this is analogous to the point spread function for the algorithm, but it must be noted that because of the reset step, the shape of this function varies depending on the image content. From left to right we have: input image, McCann99 4-iteration output, Frankle-McCann 4-iteration output.*
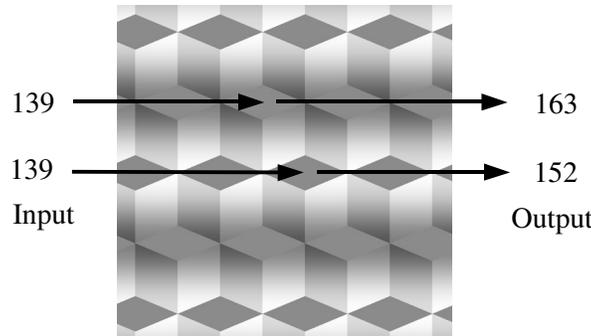


*Figure 3. Logvinenko cubes pattern illusion. As shown on the left, the input values of the cube tops are equal despite the fact that we see them as unequal. McCann99 4-iteration retinex output values are shown on the right.*

| Measure log 10 image | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Min | | | | | | | | | | | Max |
| Object in image | Shade | | | | | | | | | | | |
| | Black | | | | | | | | | | | White |
| Radiance | 1 | | | | | | | | | | | 3162 |
| log10 Radiance | 0.00 | | | | | | | | | | | |
| 256 equal log | **0.00** | **0.01** | **1.13** | **1.58** | **1.70** | **1.80** | **2.10** | **2.40** | **2.70** | **3.01** | **3.31** | **3.50** |
| 256 equal ratios | 1.00 | 1.03 | 13.35 | 37.88 | 50.35 | 62.81 | 126 | 252 | 506 | 1014 | 2032 | 3162 |
| Digit | 0 | 1 | 82 | 115 | 124 | 131 | 153 | 175 | 197 | 219 | 241 | 255 |
| Convert linear 8-bit image to log10 | | | | | | | | | | | | |
| | Min | | | | | | | | | | | |
| Object in image | Shade | | | Sun | | | | | | | | |
| | Black | | Lt.Gray | Black | | Dk. Gray | | | Mid gray | | | White |
| Radiance | 1.00 | | | | | | | | | | | 3162 |
| 256 equal differences | 1.00 | | 13.40 | 38.19 | 50.59 | 62.99 | 125 | 249 | 497 | 993 | 1985 | 3162 |
| log10 Radiance | **0.00** | | **1.13** | **1.58** | **1.70** | **1.80** | **2.10** | **2.40** | **2.70** | **3.00** | **3.30** | **3.50** |
| digit | 0 | | 1 | 3 | 4 | 5 | 10 | 20 | 40 | 80 | 160 | 255 |

*Table 1. The data in this table describes the care one must take in preparing input images. The data comes from the image of two test targets: one in sun, the other in shade (See Figure 1). The top box demonstrates the digitization of raw image data as equally spaced $log_{10}$ increments. The bottom box demonstrates the digitization of raw image data as equally spaced 8-bit linear increments, subsequently converted to $log_{10}$. Equal log increments divide the image into equal ratio steps of 1.0321, while equal linear increments divide it into equal radiance differences of 13.3971. The ratio increments represent the image well, while the linear increments severely truncate the meaningful data. The first and last columns show the dynamic range of the scene. The "Object in image" row identifies that the minimum (Min) corresponds to the black patch in the shade and the maximum (Max) that corresponds to the white patch in the sun. In both boxes, minimum relative radiance is 1.0 with a $log_{10}$ radiance of 0.00 and a digit value of 0. Maximum radiance is 3162.28, $log_{10}$ radiance is 3.50, and has a digit value 255. In the top box, the row "256 equal log" digitizes input radiances into 256 equal log increments .The row "256 equal ratios" converts this back to linear values. Column "Min" shows the starting point. The next column shows that digit 1 corresponds to 0.01 $log_{10}$ and 1.03 linear. In the bottom box, the row 256 equal differences digitizes input radiances into 256 equal linear increments. The row "log10 Radiance" converts this to $log_{10}$ values. Column "Min" shows the starting point. The third column shows that digit 1 corresponds to 1.13 $log_{10}$ and 13.40 linear. Looking up to the top box shows that the 1.13 $log_{10}$ corresponds to digit 82. The top box divides 1 linear digit into 82 $log_{10}$ digits. The linear digits compress all the shade information from light gray to black into 1 digit. The remaining columns match $Log_{10}$ radiance values and demonstrate the very poor use of digits when converting 8-bit linear digits to log digits.*

Figure 3 shows Logvinenko's gradient experiment which generates a large lightness change between the diamonds. A vertical sinusoidal gradient in non-diamond areas creates the illusion. The numbers on the left side of Figure 3 show that the input digits for the light and the dark diamond faces are both 139. The numbers on the right show the output from the corresponding faces to be 152 and 163 after McCann99 4-iteration processing. McCann[20] reports that "Retinex models can predict appearances that were previously attributed to cognitive behavior." Figure 4 shows pseudocolor renditions of input (left) and output (right) of the Logvinenko illusion. The diamond shaped tops of the cubes are equal on the left and unequal on the right. It is interesting to note, that the upper faces of the output cubes are not uniform.

Figure 5 shows the effect of McCann99 applied to a color image with a substantial blue color cast. The algorithm has been applied to each of the color channels independently. Clearly, in this case the color cast has been removed. Retinex differs from many color constancy methods in that it does not aim to find a single chromaticity for the scene illumination as is the case, for example, in the neural network [6] and color by correlation [4] methods. Retinex instead adjusts the image colors in a non-global manner as is necessary since the model attempts to match human visual response. Some effects of this can be seen in the way that in Figure 5B some of the green bleeds into the white area surrounding the "C" in "Compiler," and the way the blue is darkened near the white lettering on the right-hand blue book.

## Conclusions

We have presented new, very concise Matlab implementations[2] of two of the main practical retinex algorithms. Our hope is that this will eliminate much of the variability in what is meant when different researchers refer to retinex and thereby facilitate further rigorous testing and discussion of the method. For modeling human vision these Matlab programs depend on calibrated input data. Although these Matlab programs provide the details of how pixels are compared and processed during the ratio-product-reset-average steps of retinex processing, they do not provide details on the selection of an appropriate postlut for a particular output device. The postlut must be provided by the reader.

## Acknowledgements

## References and Notes

1. Barnard, K. and Funt, B., "Investigations into multi-scale retinex (MSR)", in *Colour Imaging Vision and Technology*, ed. L. W. MacDonald and M. R. Lou, pp.17-36, John Wiley, 1999.

2. Brainard, B., and Wandell, B., "Analysis of the retinex theory of color vision", *JOSA Vol3. No10*, Oct 1986.

3. Campbell, F. W., and Robson, J. G., "Application of Fourier Analysis to the visibility of Gratings", *J. Physiol,* (Lond) 197, 551-566 1968.

4. Finlayson, G., Hubel, P., and Hordley, S., "Color by Correlation", *Proc. IS&T/SID Fifth Color Imaging Conference: Color Science, Systems and Applications*, 6-11, Scottsdale 1997.

5. Frankle, Jonathan and McCann, John, "Method and Apparatus for Lightness Imaging", *US Patent #4,384,336,* May 17, 1983.

6. Funt, B., Cardei, V. and Barnard, K., "Learning Color Constancy", *Proc. IS&T/SID Fourth Color Imaging Conference: Color Science, Systems and Applications*, 58-60, Scottsdale, Arizona, November 1996.

7. Jobson D. J., Rahman Z., Woodell G. A.; "A Multi-Scale Retinex For Bridging the Gap Between Colour Images and the Human Observation of Scenes", *IEEE Transactions on Image Processing: Special Issue on Colour Processing*, July 1997.

8. E.H.Land, "Colour in the Natural Image", *Proc. Roy. Institution Gr. Britain*, 39, 1-15, 1962.

9. Land, Edwin and McCann, John, "Lightness and Retinex Theory", *Journal of the Optical Society of America,* 61(1), January 1971.

10. Land, Edwin and McCann, John, "Method and System for Image Reproduction Based on Significant Visual Boundaries of Original Subject", *US Patent 3,553,360*, 1971.

11. Land, E. H., Ferrari, L., Kagan, S. and McCann, J., "Image Reproduction System which Detects Subject by Sensing Intensity Ratios", *United States Patent 3,651,252*, 1972.

12. Land, Edwin, "The Retinex Theory of Color Vision", *Scientific American*, 237, 108 - 128, 1977.

13. Land, Edwin, "An alternative technique for the computation of the designator in the retinex theory of color vision", *Proc. of the National Academy of Science USA,* 83, 2078-3080, May 1986.

14. Marini, D., and Rizi, A., "Color Appearance Approach to Image Database Visual Retrieval", *Proc. SPIE 3964*, G. Beretta (ed.) 186-195 San Jose, 2000.

15. McCann, J., Savoy, R.L., and Hall, J., and Scarpetti, J., "Visibility of Continuous Luminance Gradients", *Vision Research*, 14, 917-927, 1974.

16. McCann, John, McKee, Suzanne and Taylor, Thomas, "Quantitative Studies in Retinex Theory, A Comparison between Theoretical Predictions and Observer Responses to the Color Mondrian Experiments", *Vision Research,* 16, pp. 445-458, 1976.

17. McCann, J., and Houston, K.L., "Color Sensation, Color Perception and Mathematical Models of Color Vision", in: *Colour Vision*, (ed.) Mollon, J.D. and Sharpe, Academic Press, London, 891-894 1983.

18. McCann, J., "Calculated Color Sensations applied to Color Image Reproduction", *SPIE Vol. 901 Image* Processing, Analysis, Measurement, and Quality, Bellingham, WA 205-214, 1988.

19. McCann, J., "Color Mondrian Experiments Without Adaptation", *Proceedings AIC,* Kyoto, 159 –162, 1997.

20. McCann, John, "Lessons Learned from Mondrians Applied to Real Images and Color Gamuts", *Proc. IS&T/SID Seventh Color Imaging Conference,* pp. 1-8, 1999.

21. Rabbani, M. and Jones, P.W., "Digital Image Compression Techniques", *SPIE Press*, Bellingham, 1991.

22. Savoy, R.L., "Low Spatial Frequencies and Low Number of Cycles at Low Luminances", *J. Photogr. Sci Eng.*, 22, No. 2, 76-79, 1978.

23. H. Wallach, *J. Exptl. Psychol*, 38, pg. 310, 1948.

24. Wray, W. R., "Method and apparatus for image processing with field portions", *United States Patent 4,750,211* 1988.

Legend



255  248  239  232  223  216  207  200  191  184  175  168  162  152  149  147  145  142  120  104  95  88

*Figure 4. Pseudocolor representation of a portion of the Logvinenko cubes input (left) and McCann99 4-iteration output (right). Note that despite the fact the upper cube faces on alternating rows appear to differ in intensity, the top faces of all the cubes are in fact both uniform and equal. In the output, however, the top faces of the cubes are no longer equal nor are they completely uniform.*



*Figure 5B*



*Figure 5A*



*Figure 5C*

*Figure 5: A: Input with blue color cast created by scene illumination for which the camera was not balanced. The image also has extended dynamic range obtained by frame averaging. B: Output from the McCann99 4-iteration. C: Output from the Frankle-McCann 4-iteration. The results here can be compared with those of Barnard [1]. Note that both the input and output images have been adjusted with postluts for printing.  The actual retinex input image is in log space.*